The Impact of Chaining Convolutional and Recurrent Architectures for the Task of Question Classification

Katherine Krajovic
Department of Computer Science
Brandeis University
Waltham, MA, USA
krajovic@brandeis.edu

Abstract

This paper describes an investigation into the effectiveness of chaining CNNs and LSTMs for the task of question classification. It shows that in all cases model chaining improves results over those obtained with baseline models, but that the order of this chaining does have an impact on performance. Specifically, a joint CNN-LSTM model performs better than a joint LSTM-CNN model on this task, even with identical parameters and setups. The results presented here emphasize the need for controlled comparative studies if we are to understand and make claims about the impacts of chaining these architectures.

1 Introduction

Question classification is a critical step in question answering systems which aim to provide accurate answers to natural language questions. As a part of this process questions are classified according to the type of answer they require. For example, the question "who was the first president of the United States?" requires that the answer be a person. This categorical information can be used to improve the performance of a question answering system.

This problem has been thoroughly researched and approached with many different techniques (Cortes et al.). Many different models utilizing convolutional and recurrent methods have been proposed, including models that chain these two architectures together. This paper aims to analyze the impact of this chaining using the standard TREC question classification dataset from Li and Roth (2002).

2 Background

Zhou et al. (2015) present a C-LSTM model in which they extract important features using a convolutional layer and then pass these into a recurrent layer which can extract long distance dependencies across the input question. They report an accuracy of 94.6% on the TREC question classification dataset, which is commonly used to evaluate performance on this task. In a different approach Zhang et al. (2016) use an LSTM layer to generate a representation of the input that captures long distance dependencies as a first step, and then send the output of this layer into a CNN to extract the most important features. They report an accuracy of 95.6% on the TREC question classification dataset.

While these studies both present results on a standard dataset, the differences in model parameters, setups, and unknown data processing techniques prevent conclusions from being drawn on the impact of the ordering of these two architectures.

The present study aims to investigate this ordering by creating baseline CNN and LSTM models tuned to maximize accuracy on this task and then, with identical data processing techniques and model parameters, chaining these two architectures and analyzing the impact on results.

3 Data Processing

This section describes the dataset used and the steps taken to prepare the data for use in the models.

3.1 Dataset

As referenced above, the dataset used in this study was the standard TREC question classification dataset from Li and Roth (2002). It consists of natural language questions whose answers belong to one of Abbreviation-ABBR, Description-DESC, Entity-ENTY, Human-HUM, Location-LOC, Number-NUM. The questions are further classified into 50 fine-grained categories, but for the purpose of this investigation we classify based only on the six top-level categories. The training set contains 5,382 questions, and there are an additional 489 questions in a held out test set.

3.2 Preprocessing

The data are already tokenized, with a single space separating each token. Certain punctuation and morphemes are considered independent tokens, for example: *:*, *;*, *'s*, ,(comma), *"*, *'*, and *?*. Hyphenated words are not split, and . is left attached to the end of titles (Dr. Mr. Mrs.) and acronyms (D.C., U.S).

Each question is represented as a list of these tokens. No further tokenization or normalization of the tokens was performed. A vocabulary was generated from all tokens in the training set. This vocabulary was then used to transform the input questions into a list of indices which was used for pre-trained embedding lookup in the first step of each model. Any tokens that did not appear in the vocabulary of the pre-trained embeddings were initialized with a random vector.

The questions were padded or truncated to a maximum length (default 30). For any question with fewer tokens than the expected number of tokens, a *pad* token was appended until the token list reached the desired length. Any question with more than the maximum number of tokens was truncated.

4 Models

The following sections describe the structure of each of the four models that are presented in this study. These include a standard CNN and LSTM to serve as baselines, and a joint LSTM-CNN model and CNN-LSTM model created by chaining the two baselines together.

The first step in each of these models was embedding lookup in an embedding layer initialized with pre-trained word embeddings. The final step was a dense layer to narrow the representation to a vector of length six (the number of possible labels), over which softmax could be taken to find the predicted label.

4.1 CNN

Several different configurations were experimented with for the CNN model. These included performing convolutions concurrently or in sequence, with different numbers of kernels, different kernel widths, and different numbers of convolutional layers.

Ultimately the model that yielded the highest accuracy was created by performing a single convolution for each input kernel width in parallel and then concatenating the results of these into a single representation. Concatenation along the dimension of sequence length (horizontal) and concatenation along the embedding or filter size dimension (vertical) were both experimented with. Global average pooling was applied to the output of this, and the result was then fed into a final dense layer.

While both forms of concatenation resulted in reasonable performance on this task, the horizontal concatenation produced an output that was not suitable for passing into an LSTM, as the LSTM expects a sequence, not two concatenated representations of a sequence. This made the vertical concatenation approach more useful for the comparison that this study aims to make because it resulted in an output that maintained a similar shape and order of the input sequence. In order to enable this vertical concatenation, the outputs of each convolution needed to be the same length. The output length could be controlled to some extent through padding, but padding could only be performed on both sides of the input. Because of this, viable kernel widths for the model with vertical concatenation were restricted to those which were two units apart from each other, for example 2,4, or 3,5 but not 2,3. This limitation was the motivation for experimenting with both the vertical and horizontal concatenation models, even though the horizontal concatenation model could only be used in the joint LSTM-CNN model and not the CNN-LSTM.

4.2 LSTM

The LSTM model was a simple bidirectional LSTM with the length of the hidden units equal to the dimension of the input embeddings. The output of the encoding step was fed through a global average pooling layer and finally to the dense output layer.

4.3 LSTM-CNN

The LSTM-CNN model consisted of an embedding layer and encoding layer identical to those in the base LSTM model. The final hidden state from the encoder (before pooling) was passed into a set of convolutional layers identical to those in the base CNN model. Both the vertical and horizontal concatenation methods of the outputs of the convolutional layers were experimented with. For each of these the output was then passed through a global average pooling layer and the final dense layer.

Kernels	Filter Size	Dev Accuracy
3,4	100	0.831
2,3	100	0.843
2,3,4	100	0.836
2,3	80	0.840
2,3	60	0.840
2,3	40	0.836
2,3	20	0.824

Table 1: CNN: Convolutional Layer Tuning

4.4 CNN-LSTM

The CNN-LSTM model began with the convolutional layers described in the base CNN. For this model, only the vertical concatenation method for the CNN output was viable because the sequential nature of the input needed to be preserved in order to be fed into the LSTM. The output of the CNN was then used as input to an LSTM encoding layer. The output was fed through a global average pooling layer and into the final dense layer.

5 Baseline Model Tuning

This section describes the process of finding parameters to maximize accuracy of the baseline models.

5.1 Cross Validation

Because the dataset for this study is relatively small, k-fold cross validation was used for all hyperparameter tuning and testing. Specifically we used five folds, with four of the folds used as the training set and the fifth fold used as a held out development set for each of the five iterations of a test. The recorded accuracy for any given set of parameters was the average of the accuracies obtained across the five iterations.

5.2 CNN Tuning

The initial CNN tuning was performed on the model that makes use of horizontal concatenation of the outputs of the parallel convolutional layers. Table 1 shows the results of experimentation with the parameters of the convolutional layers themselves. It can be seen that two convolutions, one with a kernel size of 2 and the other with a kernel size of 3, along with a filter size of 100 produced the best results. These kernel sizes were then carried forward into the experimentation shown in Table 2. It was observed that 300 dimensional Glove embeddings produced the highest accuracies. In Table 5, we can see that the optimal sequence length was 15.

Embeddings	Filter Size	Dev Accuracy
Glove 200	100	0.842
Glove 200	150	0.843
Glove 200	200	0.842
Glove 300	150	0.844
Glove 300	200	0.844
Glove 300	300	0.844

Table 2: CNN: Embedding and Filter Size Tuning

Model	Dev Accuracy
Horizontal	0.842
Vertical	0.850

Table 3: CNN: Vertical vs. Linear Concatenation

Finally, Table 3 displays the difference between the vertical and horizontal concatenation methods. For the sake of an even comparison, both of these tests were run with two kernel sizes, 2 and 4, and all fine tuned hyperparameters according to the experiments discussed above. The only difference was the method of concatenation, and it can be seen that the vertical concatenation performs better.

5.3 LSTM Tuning

There were fewer parameters to tune with the base LSTM model. Table 4 shows that 300 dimensional Glove embeddings produced the highest accuracy. It can be seen in Table 5 that, similar to the CNN model, a maximum sequence length of 15 produced the best results.

6 Results

This section presents the results on the development and test sets for all comparable models.

6.1 LSTM-CNN Results

Table 6 shows the results for the LSTM-CNN models using 5-fold cross validation on the training data as mentioned above, and Table 7 presents the results on the test data. It can be seen that the vertical concatenation LSTM-CNN outperforms the

Embeddings	Dev Accuracy
Glove 100	0.779
Glove 200	0.805
Glove 300	0.810

Table 4: LSTM: Embedding Tuning

Model	Sequence Length	Dev Accuracy
CNN	35	0.842
CNN	30	0.844
CNN	25	0.846
CNN	20	0.848
CNN	15	0.849
LSTM	30	0.810
LSTM	25	0.810
LSTM	20	0.820
LSTM	15	0.821

Table 5: Tuning Sequence Length

horizontal concatenation LSTM-CNN models in both cases, by 0.013 on the training/development data and by 0.008 on the test data.

It is most important to note that both LSTM-CNN models, horizontal and vertical, outperformed the baseline LSTM model by 0.035 and 0.043 respectively.

6.2 CNN-LSTM Results

Again, Table 6 shows the performance of the CNN-LSTM model using 5-fold cross validation on the training data. This model outperforms the baseline CNN (recall that only the vertical concatenation method is comparable here) by 0.005. Similarly, on the test data, the CNN-LSTM outperforms the baseline CNN by 0.014. The only instance of a decrease in performance between a baseline and chained model was that the average precision for the CNN-LSTM model was lower than that of the baseline CNN model.

7 Discussion

There are several interesting observations to be made from these results. Firstly, the vertical concatenation method for the CNN produces better results than the horizontal concatenation method, even though this limits viable kernel widths to a more tightly constrained set. This can be seen in both pairs of CNN and CNN-LSTM results, with the vertical method producing higher accuracies in all cases.

In both cases of chaining, whether that was adding a CNN to the output of an LSTM or vice versa, the joint X-Y model produced higher accuracies than the X model on its own. This suggests that there are benefits to this kind of model chaining.

It is also worth noting that the baseline CNN model produced substantially higher accuracies

Model	Dev Accuracy
LSTM-CNN (horizontal)	0.831
LSTM-CNN (vertical)	0.844
CNN-LSTM	0.855

Table 6: Chained Models: Results on Dev

Model	Avg. Precision	Acc.
CNN (horiz.)	0.866	0.882
CNN (vertical)	0.895	0.900
LSTM	0.845	0.851
LSTM-CNN (horiz.)	0.864	0.886
LSTM-CNN (vertical)	0.867	0.894
CNN-LSTM	0.876	0.914

Table 7: Results on Test Data

than the LSTM baseline, a difference of 0.049. Similarly, the CNN-LSTM model produced higher accuracies than the LSTM-CNN model, but in this case with a difference of only 0.020. Attaching the CNN to the output of the LSTM resulted in a 0.043 increase, which is very similar to the original 0.049 difference between the models. Attaching the LSTM to the CNN resulted in a smaller increase. This shows that the order of the chaining is significant. It is not the case that using these two models together in any order produces the same results.

8 Conclusion

This investigation has shown that chaining convolutional and recurrent architectures, regardless of the order of this chaining, is an effective method for improving the accuracy of question classification. While chaining in either direction will improve results as compared to a baseline model, the order of the chaining does impact the performance of the model overall. In this investigation, utilizing a convolutional architecture to identify important features and then passing these to a recurrent architecture which could identify long distance dependencies produced the highest accuracy, at 91.4%. Interestingly this is contrary to the results presented by Zhou et al. (2015) and Zhang et al. (2016) in which the LSTM-CNN performed better. This emphasizes the importance of performing controlled studies if we are to make claims about the effectiveness of linking these models.

References

Eduardo Cortes, Vinicius Woloszyn, Arne Binder, Tilo Himmelsbach, Dante Barone, and Sebastian Möller. An empirical comparison of question classification methods for question answering systems.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

Rui Zhang, Honglak Lee, and Dragomir Radev. 2016. Dependency sensitive convolutional neural networks for modeling sentences and documents. *arXiv preprint arXiv:1611.02361*.

Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. 2015. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*.